A red and black logo

Description automatically generated

MSc in Data Science

Project Report

2025/26

Enhancing Image Captioning using ConvNeXt with LSTM and Transformer Decoders

Sajeel Nadeem Alam

Supervised by: Dr. Kevin Ryan

Date of Submission

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: Sajeel Nadeem Alam

# Abstract

Keywords:

# Table Of Contents

[Abstract 2](#_Toc206631277)

[Table Of Contents 3](#_Toc206631278)

[1. Introduction and Objectives 5](#_Toc206631279)

[2. Context 5](#_Toc206631280)

[3. Methods 5](#_Toc206631281)

[3.1. Dataset and Preprocessing 5](#_Toc206631282)

[3.1.1. Dataset 5](#_Toc206631283)

[3.1.2. Preprocessing 5](#_Toc206631284)

[3.1.3. Dataloader 6](#_Toc206631285)

[3.2. Model Architecture 6](#_Toc206631286)

[3.2.1. ConvNeXt Encoder 6](#_Toc206631287)

[3.2.2. Baseline Decoder: LSTM with Attention 7](#_Toc206631288)

[3.2.3. Proposed Decoder: Transformer Decoder 7](#_Toc206631289)

[3.3. Experimental Design 8](#_Toc206631290)

[3.3.1. Training Strategies 8](#_Toc206631291)

[3.3.2. Finetuning ConvNeXt 9](#_Toc206631292)

[3.3.3. Decoding Strategy 10](#_Toc206631293)

[3.3.4. Word Embeddings 10](#_Toc206631294)

[3.4. Training and Hyperparameters 10](#_Toc206631295)

[3.5. Evaluation Metrics 11](#_Toc206631296)

[4. Results 12](#_Toc206631297)

[4.1. Preliminary Results 12](#_Toc206631298)

[4.1.1. Results on Flickr8k 12](#_Toc206631299)

[4.1.2. Results with Original Study’s Codebase 13](#_Toc206631300)

[4.2. Baseline Performance on MS COCO 13](#_Toc206631301)

[4.2.1. The Impact of Training Strategies 13](#_Toc206631302)

[4.2.2. Selecting the Best Decoder 15](#_Toc206631303)

[4.3. The Impact of Finetuning ConvNeXt 16](#_Toc206631304)

[4.3.1. Quantitative Analysis 16](#_Toc206631305)

[4.3.2. Qualitative Analysis 17](#_Toc206631306)

[4.4. The Impact of Pretrained Word Embeddings 17](#_Toc206631307)

[Discussion, Reflection and Conclusion 18](#_Toc206631308)

[Glossary 18](#_Toc206631309)

[References 18](#_Toc206631310)

[Appendices and additional files 18](#_Toc206631311)

# 1. Introduction and Objectives

# 2. Context

# 3. Methods

This chapter goes through the experimental and technical setup of the image captioning system developed in this study. The methodology explains the dataset used, the pipeline built to load it into the model, the baseline and proposed model architectures, the experiments conducted and the evaluation metrics used.

## 3.1. Dataset and Preprocessing

### 3.1.1. Dataset

The dataset used in this study is the MS COCO dataset which was originally made by Microsoft for the COCO 2015 Image Captioning challenge and has since been used widely by researchers for the image captioning task, making it a standard benchmark in the community and allowing for meaningful comparisons of this study’s results with existing works. In the initial stages of the study, the smaller Flickr8k dataset was used to test the robustness of the architecture with fewer number of images. For both datasets, the popular Karpathy split (Karpathy & Fei-Fei, 2017) was used according to which the MS COCO dataset has 123,287 images out of which 5000 are for validation, 5000 are for testing and the rest are for training whereas the Flickr8k dataset has 8000 images out of which 1000 are for validation, 1000 are for testing and the rest are for training. Each image is well-annotated with at least five captions providing variety for evaluation.

### 3.1.2. Preprocessing

The image and caption data were preprocessed using multiple steps for efficient data loading and model training. Firstly, the Karpathy split file is loaded as a JSON which contains image metadata and a list of captions for each image, already organized into training, validation and testing splits. Using this file, all the captions and their respective images are iterated over and captions with lengths greater than 50 are discarded, followed by splitting the images and the captions into training, validation and testing. A vocabulary, or a word map is made in which each word is mapped to a unique index and words appearing less than 5 times across all the captions are not included to mitigate the effect of rare words. Four special tokens such as <unk>, <start>, <end> and <pad> are added to the vocabulary with their own unique IDs. The word map is serialized as a JSON file to be used consistently across model training and inference.

The core of the preprocessing involves converting raw .jpg images and captions into a format suitable for the model. For each split (train, validation and test) three files are generated. The first file consists of image data for which all the images in that split are loaded, converted to RBG, resized to 256x256 pixels and stored in a single .HDF5 file. This file format allows for efficient loading of image data which is essential for training using large datasets. The second file consist of all the captions stored in JSON format. For each image, its five captions are sampled and then encoded into numerical sequences using the generated word map. Each sequence is prepended with the <start> token, appended with the <end> token and then padded with the <pad> token until it reaches a maximum caption of length of 50. The third file which is also a JSON stores the true length of each caption for every image. This preprocessing ensures that the image and caption data are aligned, standardized and optimized to be used efficiently for training and inference.

### 3.1.3. Dataloader

To ensure efficient loading of data from the .HDF5 and JSON files, a custom PyTorch dataset class is set up. This dataset class ensures that the large .HDF5 file containing all the image data is not loaded altogether at the time of initialization. Instead, lazy loading is implemented which opens the .HDF5 file only once at the time of first data retrieval and the subsequent images are retrieved only when required. This approach prevents out of memory (OOM) issues from loading the entire image data at once. The smaller captions and caption lengths JSON files are loaded fully at the time of initialization for faster access. The dataset is configured to handle the structure of MS COCO where each image has five captions and ensures that every caption is returned with its corresponding image. This is important for validation and testing since every image requires all five of its captions to calculate the evaluation metrics.

This dataset is passed into PyTorch’s dataloader which manages the loading of data in batches to feed the model. In this study, the batch size is 32 so the dataloader loads 32 images, captions and their lengths together in one batch. The training data is shuffled at the start of each epoch to prevent the model from learning the order of the data. The dataloader performs multi-process data loading with 6 worker processes allowing the CPU to pre-fetch data while the GPU is computing leading to faster training. These worker processes are kept alive throughout the training process to reduce the overhead of re-initializing them and opening the .HDF5 file again at the start of each epoch. These careful data loading and batching strategies are crucial for loading data efficiently thus speeding up the training and inference process.

## 3.2. Model Architecture

### 3.2.1. ConvNeXt Encoder

As mentioned earlier, this study uses an encoder-decoder architecture in which the encoder is a ConvNeXt. The ConvNeXt is an improved CNN built by the research team at Meta which builds on the foundational ResNet model and incorporates features from vision transformers. Upgrades such as using larger convolution kernels allow the model to gather broader context across the image while a hierarchical feature learning approach inspired from Swin Transformers processes the image at different resolutions. This enables the model to capture fine-grained local features at lower resolutions combined with broader context at higher resolutions and integrate them in the feature vectors (Liu et al., 2022). This study uses the base version of ConvNeXt from PyTorch which is pretrained on the ImageNet-1k dataset. It has a feature extractor layer followed by a pooling layer and a classification head. For the use case of image captioning, the pooling layer and classification head are removed since ConvNeXt is used only to extract image features. The feature extractor of ConvNeXt has seven sequential layers, each containing several convolutional layers.

Images are passed as batches to the ConvNeXt. The output of the feature extractor is a high-dimensional feature map where the channel dimension is 1024. This is passed through an adaptive average pooling layer which resizes the feature map to a fixed size of 7x7 which is permuted to the shape (batch size, 7, 7, 1024). This ensures that the ConvNeXt encoder can accept images of varying size but produces consistent sized output tensors that are compatible with the decoder. A key aspect of the encoder is the ability to freeze certain layers while fine-tuning preventing any updates to the pretrained weights. Initially, all the layers in the feature extractor are frozen to get a baseline performance and in later experiments, certain layers are unfrozen to check the effect on the performance of the architecture. This flexibility allows the study to empirically investigate how fine-tuning till certain depths of the encoder affects the model’s overall performance.

### 3.2.2. Baseline Decoder: LSTM with Attention

In the baseline model, the decoder is an LSTM with an integrated attention module. The decoding process begins by averaging the ConvNeXt’s image features to get the initial hidden and cell states for the LSTM and their dimension is set to 512. At each time step, the LSTM’s current hidden state is passed to an attention module along with the encoder’s image features. The attention module learns to compute a relevance score for each image feature based on the hidden state. The scores are normalized using a softmax function to get attention weights or alpha scores which are multiplied element-wise to the encoder’s image features and then summed to get an attention-weighted context vector which highlights the most relevant regions of the image to predict the next word. The weighted context vector is passed through a gating mechanism based on the hidden state which allows the model to reduce its focus on features that it has already described.

The gated attention-weighted context vector is concatenated with the embeddings of the previous word which is embedded using a standard embedding layer from PyTorch and also has a dimension of 512. In the case of teacher forcing, this word is the ground truth and in the case of no teacher forcing, this word is the model’s own prediction from the previous step. This concatenated vector is fed into a LSTM cell along with the hidden and cell states to return updated hidden and cell states. The updated hidden state is passed through a fully connected layer to get the logits across the entire vocabulary. These logits are used to calculate the loss during training or generate the next word for inference. This iterative process continues to calculate the prediction scores across the vocabulary for each next word in the caption until the <end> token is generated or the maximum caption length is reached.

### 3.2.3. Proposed Decoder: Transformer Decoder

This study proposes a transformer decoder to replace the LSTM. The inherent parallel processing ability of the transformer considers all regions of the image and the generated caption simultaneously, providing global context during the caption generation process (Liu et al., 2021). The true encoded captions are embedded using a standard embedding layer and have a dimension of 512. Since the decoder processes all the words in parallel, it is not aware of the position of each word in the sequence hence the caption embeddings are passed through a positional encoding module which incorporates sine and cosine signals in the embeddings to provide the model with positional information for each word.

The core of the decoder is a stack of six transformer decoder layers and each layer has eight heads for multi-headed attention. The positional encoded embeddings of the caption along with the image features from the encoder are passed into the transformer decoder which contains two attention mechanisms. The masked multi-headed self-attention allows the decoder to attend to all the previous words in the caption sequence. A mask is applied to prevent the decoder from looking at the future words while generating the next word. The multi-headed cross-attention enables the decoder to create a visual context vector using the image features which makes it focus on the relevant parts of the image while generating the next word. In the case of teacher forcing, the true caption is embedded followed by positional encoding before being fed into the transformer decoder. In the case of no teacher forcing, the caption is generated word by word instead of in parallel and in order to generate the next word, all the previously generated words are embedded, positionally encoded and then fed into the decoder along with the image features. In the case of greedy search, the word with the highest score is appended to the generated caption which acts as the updated input for the next step. This iterative process continues until the <end> token is generated or the maximum caption length is reached.

In both cases, the output from the transformer decoder is a sequence of feature vectors each with a dimension of 512, with the sequence being the length of the maximum caption length. This output is passed through a fully connected layer which maps these features to logits across the vocabulary for each word position in the caption length. These logits are used to calculate the loss during training or generate the next word at the time of inference.

## 3.3. Experimental Design

This section outlines the different experiments and strategies that were used to analyze the architecture’s performance. It is important to note that the encoder and LSTM decoder is inspired from the codebase (Ramos et al., 2024) so that it can be compared with the transformer decoder in this study.

### 3.3.1. Training Strategies

The models were trained using both teacher forcing and non-teacher forcing training strategies in the decoder to implement what was done in the study (Ramos et al., 2024). It is important to note that each image has five true captions and the model generates a caption for every true caption hence five captions are generated for each image. In the case of teacher forcing, previous words of the true caption are provided to the decoder to generate the next word. In non-teacher forcing the model’s own previously generated words are used to generate the next word making it an autoregressive approach.

However, while replicating the original codebase it was discovered that although their study stated that the model was trained without teacher-forcing as well and in fact performed better, their codebase only trained the model with teacher forcing. Moreover, at the time of validation and testing, inference was done using teacher forcing in the decoder which is not correct since at inference time it is assumed that the model does not have access to true captions. In order to address these issues and provide a robust comparison, this study first implements teacher forcing for both decoders during training and inference just to replicate what was done in the original study. Then non-teacher forcing is implemented from scratch and in separate experiments both the LSTM and transformer decoders are trained with and without teacher-forcing, and inference at the time of validation and testing is done the correct way without teacher-forcing. Their results are compared and the decoder + training strategy with the highest score on the test set was selected for further experiments.

### 3.3.2. Finetuning ConvNeXt

In the original study (Ramos et al., 2024), the authors chose not to fine-tune the ConvNeXt and relied on the pre-trained weights. However, a key area of investigation in this study is to evaluate how the depth of fine-tuning the pre-trained ConvNeXt during training affects the overall performance. Earlier layers are responsible for more general features such as edges and lines whereas later layers recognize more complex patterns that are specific to the task making this an interesting investigation. The hypothesis is that fine-tuning deeper layers will yield better results. As mentioned earlier, the ConvNeXt has 7 sequential layers. In the experiments where the encoder was fine-tuned, it was frozen for the first 20 epochs to allow the gradients of the decoder to reach some stability and avoid corrupting the pre-trained weights of the ConvNeXt with large initial updates. There were four main scenarios when it comes to fine-tuning the ConvNeXt in this study as shown below:

1. **Frozen encoder - no fine-tuning**: All the layers of the ConvNeXt were frozen and no fine-tuning was done to replicate the original study (Ramos et al., 2024). Both LSTM and transformer decoders were trained with teacher forcing and non-teacher forcing with the frozen ConvNeXt. The decoder + training strategy which performed the best in this scenario was then used for further experiments.
2. **Fine-tuning layers 5-7:** The last three layers of the ConvNeXt were fine-tuned. The initial experiment in this scenario used a learning rate of 1×10-4 with a patience of 20 epochs for early stopping. To investigate the effects of a more gradual convergence, two additional experiments were conducted using lower learning rates of 1×10-5 and 1×10-6, both with an increased patience of 40 epochs.
3. **Fine-tuning layers 3-7:** This experiment was conducted to explore the impact of fine-tuning deeper into the network to determine if adapting intermediate layers that combine simple features into more complex patterns will result in an improvement. A learning rate of 1×10-4 with a patience of 20 epochs was used.
4. **Fine-tuning layers 1-7:** In this experiment, the entire ConvNeXt encoder was fine-tuned to explore the effects of adapting the full network. To ensure stable convergence and prevent significant alteration of the pre-trained features, a low learning rate of 1×10-6 was used with a patience of 40 epochs.

For each experiment, the model which resulted in the best performance on the validation set during training was saved as a checkpoint. The checkpoints were then tested on a test set to ensure a fair comparison of results.

### 3.3.3. Decoding Strategy

During training without teacher forcing and at the time of inference, the decoder does not have access to the true caption and relies on its own outputs to generate the next word. As mentioned earlier, at every word position the decoder calculates logit scores across all the words in the vocabulary. In order to select the next word, this study implements greedy search which selects the word with the highest logit score at every word position in the generated caption. This becomes the input for the next time step.

### 3.3.4. Word Embeddings

Another key objective of this study was to investigate the effect of prior linguistic knowledge on the quality of generated captions. For this purpose, after selecting the best performing decoder, training strategy and ConvNeXt fine-tuned layers, various word embeddings were evaluated. For the initial experiments, the decoder’s embedding layer which mapped the encoded captions to word embeddings was initialized with random weights. A dimensionality value of 512 was chosen for these embeddings and their values were learnt from scratch during the training process.

To explore the effect of external knowledge on the model’s performance, two more experiments were conducted with pre-trained word embeddings while keeping the rest of the configuration constant. The decoder’s embedding layer was initialized with vectors from both Word2Vec and GloVe models. This provided the model with a rich, pre-existing understanding of word meanings and relationships while mapping the encoded captions to word embeddings, which were then fine-tuned during training to adapt specifically to the image captioning task. The models’ performances were then compared to the one with random word embeddings.

## 3.4. Training and Hyperparameters

The models are trained for 120 epochs with an early stopping set if there is no improvement in the validation score for 20 epochs. Cross-entropy is used as the loss function. The optimizers for both the encoder and decoder are Adam. The learning rate for the decoder optimizer is 1×10-4 whereas for the encoder optimizer the values 1×10-4, 1×10-5 and 1×10-6 are tested. The learning rates are scaled down to 80% if there is no improvement for 8 epochs. After every epoch a checkpoint containing all the information to resume training from that point is saved. The best checkpoint based on the validation score is saved and updated throughout the training cycle. Once training is complete, the best checkpoint is used for testing on the test set.

In order to train the encoder-decoder architecture, images are loaded in batches of size 32 using the dataloader. For each batch, images are passed to the ConvNeXt which extracts image features for every image. These image features along with their respective captions and caption lengths are passed to the decoder. For each image, at every word position the decoder calculates a set of raw scores known as logits, across the entire vocabulary to generate a caption. The resulting logits which are the model’s prediction are compared against the ground truth word labels in the cross-entropy loss function which calculates the average loss across all the predicted tokens in the batch. In the case of the LSTM decoder, a doubly stochastic attention regularization loss is added to this total loss to encourage the model to focus on all parts of the image rather than just one area. The loss is backpropagated throughout the architecture and the optimizers for the encoder and decoder update their respective model’s parameters. To avoid exploding gradients and ensure stable model convergence, gradient clipping is applied to clamp the parameters of both the encoder and decoder optimizers to a threshold of 5. This iterative process is repeated to update the parameters of the entire architecture after every batch in each epoch.

In the case of teacher forcing, the length of the generated caption and the true caption are always the same thus the predicted logits are always aligned with the ground truth word labels to calculate the loss. To ensure that the loss calculation is performed only on relevant tokens, the padding tokens following the <end> token are excluded from both the predicted logits and the ground truth labels. In non-teacher forcing, there is a possibility that the generated caption may be shorter or longer than the true caption with a maximum caption length set at 50. To overcome this challenge, the length of the generated caption till the <end> token is calculated and used to slice the predicted logits and true caption followed by a non-padding mask to ensure that any padding tokens at the end of the predicted logits or true captions are filtered out. This aligns the predicted logits and ground truth word labels for loss calculation on only the relevant tokens.

The models were trained using two NVIDIA A-100 40GB GPUs. A multi-GPU training system was set up in which the number of batches are divided amongst the two GPUs thus speeding up the training process which was essential when dealing with large image data. Each GPU calculated its own local loss and during backpropagation the gradients of this local loss with respect to every parameter on that specific GPU were calculated. However, PyTorch’s Distributed Data Parallel package collects the gradients for all model parameters from both the GPUs and averages them before broadcasting them back to both the GPUs. This ensures that on each GPU, the model’s weights are updated with a globally consistent gradient.

## 3.5. Evaluation Metrics

The architecture’s performance was measured using a set of evaluation metrics that gave a strong insight into both the training process and the quality of the generated captions. Firstly, the efficiency of the training process was calculated using the data time which measured the time taken to load a single batch of data, and the batch time which measured the duration of a single training iteration. Both these metrics were measured for each batch and then averaged over an entire epoch.

The cross-entropy loss was calculated to give an insight into how well the model’s predicted probability distribution aligns with the actual ground truth words. A lower loss indicates a higher probability assigned to the correct word indicating a more confident model. Only the actual words in the captions were included while the padding tokens were removed. In addition to the loss, top-5 accuracy was evaluated which represents the percentage of correctly predicted words for which the true word existed in the decoder’s top five most probable predictions. A high top-5 accuracy score indicates that although the decoder might not always select the correct word, it consistently ranks the correct word very highly. Both the loss and top-5 accuracy are calculated for each batch and then averaged over an entire epoch. They are also both calculated at training, validation and test times.

At the time of inference, the quality of the generated captions is measured by the BLEU-scores. BLEU score measures how many n-grams i.e., contiguous sequence of words in the generated caption also appear in the ground truth captions. It calculates a precision score based on the overlapping n-grams from unigram to 4-gram resulting in BLEU-1, BLUE-2, BLUE-3 and BLUE-4 scores respectively. A brevity penalty is applied to captions that are too short. Higher BLEU scores indicate a greater overlap between the generated and ground truth captions which means that the generated caption is grammatically correct and has captured most of the semantic correctly. In order to calculate it, the reference (true captions) and hypothesis (generated captions) corpora are stored. The references are a nested list in which each inner list contains the five ground-truth captions for a single image, whereas the hypotheses are a single list containing five generated captions for each image. A single BLEU score is calculated by aggregating the n-gram overlap and brevity penalty over the all the tokens in the validation/test set, comparing each of the five generated captions against the set of five true captions for each image. This approach provides a robust and singular value, ranging from 0 to 1, that represents the overall quality of the model's output.

In this study, the BLEU-4 score is used to assess the performance of the architecture when checking for early stopping and saving the best checkpoint. It is also used as the primary evaluation metric when comparing models and experiments.

# 4. Results

## 4.1. Preliminary Results

### 4.1.1. Results on Flickr8k

As mentioned earlier, in this study the system is initially run on the Flickr8k dataset to test the pipeline’s robustness and models’ convergence on a smaller dataset. This helped to configure the number of workers needed in the dataloader to optimize GPU usage during training. Using the default training setup with teacher forcing in training, frozen ConvNeXt, greedy decoding at inference and decoder learning rate as 1×10-4, initial experiments are run with the LSTM and transformer decoder and the results obtained are displayed in Table 1. It is important to note that these metrics correspond to the best performing epoch according to the validation BLEU-4 score during training which is saved and tested on the test set.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Test loss** | **Test top5 acc** | **Bleu-1** | **Bleu-2** | **Bleu-3** | **Bleu-4** |
| **LSTM + Att** | 3.25 | 71.11 | 67.03 | 45.18 | 28.60 | 17.46 |
| **Transformer** | 2.76 | 71.81 | 66.45 | 44.95 | 28.98 | 18.09 |

**Table 1.** Test metrics on Flickr8k

It can be seen that the transformer decoder is a more confident model since it has a lower loss value. It also has a slightly higher BLEU-4 score suggesting that it produces captions with a greater 4-gram overlap with the ground truth captions. However, it is important to note that these results are with teacher-forcing at inference which is a flaw in the original study’s codebase (Ramos et al., 2024) and was fixed for later experiments once the pipeline’s robustness was confirmed.

### 4.1.2. Results with Original Study’s Codebase

As mentioned earlier, both decoders are trained with teacher forcing (TF) on the MS COCO dataset with inference done using teacher forcing as well to replicate the codebase of the original study. The results obtained along with the results of (Ramos et al., 2024) are displayed in Table 2. The original study only states the validation bleu scores of the best performing checkpoint during training.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Training loss** | **Validation loss** | **Bleu-1** | **Bleu-2** | **Bleu-3** | **Bleu-4** |
| **LSTM + Att (Ramos et al., 2024)** |  |  | 73.17 | 53.16 | 36.17 | 24.63 |
| **LSTM + Att** | 2.57 | 2.80 | 73.22 | 53.24 | 36.63 | 24.71 |
| **Transformer** | 1.94 | 2.14 | 73.58 | 53.96 | 37.73 | 25.86 |

**Table 2.** Training and validation metrics on MS COCO with TF at training and inference

It can be observed that the results obtained in this study are comparable to the ones in the paper (Ramos et al., 2024) showcasing that this study was able to replicate their work for comparison. The transformer decoder built in this study achieved lower loss values and slightly higher bleu scores suggesting that it is a more confident model producing accurate captions as compared to the LSTM decoder.

## 4.2. Baseline Performance on MS COCO

Once the system’s robustness is validated and the original codebase’s LSTM is replicated, this study fixes the flaws by implementing training and inference without teacher forcing. In all future experiments including these ones, inference is done without teacher forcing. In these initial experiments, the ConvNeXt is frozen to get a baseline performance and to select the best performing training strategy and decoder combination.

### 4.2.1. The Impact of Training Strategies

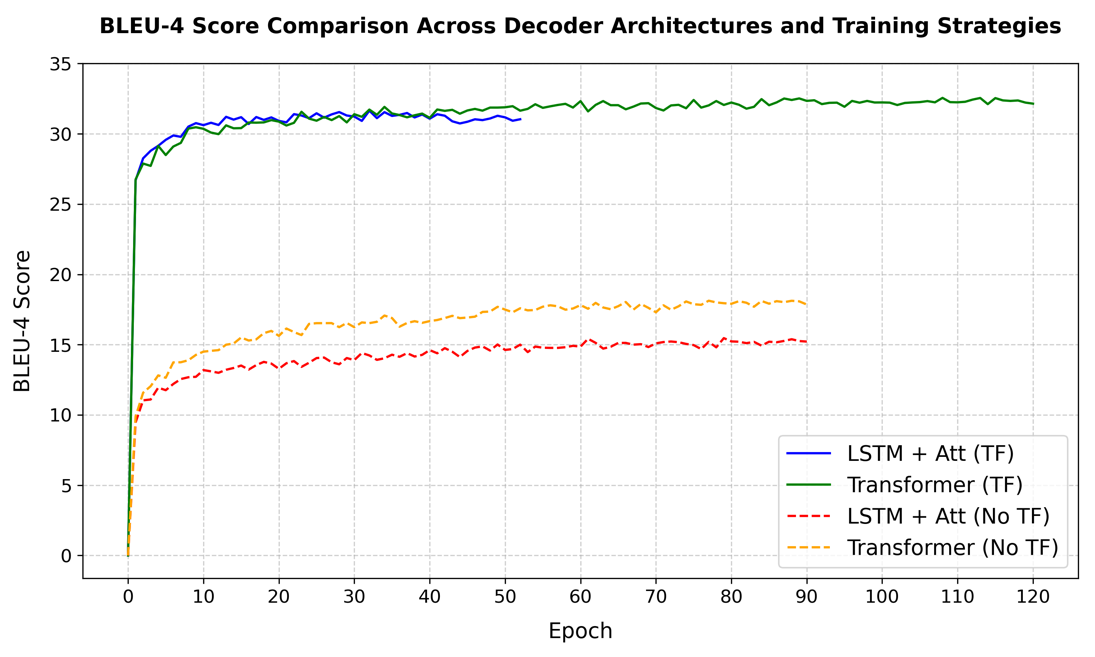
In order to investigate the impact of training strategies on the model’s performance, both the LSTM and transformer decoders are trained with teacher forcing (TF) and without teacher forcing. The training and validation evaluation metrics of the best performing checkpoint during training are displayed in Table 3 along with the best performing checkpoint from (Ramos et al., 2024).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Train loss** | **Train**  **top5 acc** | **Val loss** | **Val top5 acc** | **Bleu-1** | **Bleu-2** | **Bleu-3** | **Bleu-4** |
| **No TF** | **LSTM + Att** | 4.20 | 52.81 | 4.53 | 48.15 | 66.60 | 42.63 | 26.05 | 15.46 |
| **Transformer** | 3.48 | 51.59 | **3.85** | **48.63** | 67.63 | 45.16 | 29.03 | 18.37 |
| **LSTM + Att (Ramos et al., 2024)** | - | - | 2.78 | 77.80 | 74.79 | 58.07 | 44.61 | 34.76 |
| **TF** | **LSTM + Att** | 2.64 | 79.42 | 7.38 | 31.96 | **76.78** | **58.10** | 43.39 | 31.65 |
| **Transformer** | **1.91** | **81.03** | 8.10 | 31.61 | 76.16 | 58.01 | **43.86** | **32.56** |

**Table 3.** Training and validation metrics training with and without teacher forcing (TF)

In Table 3 it can be seen that in the case of teacher forcing for both decoders, the validation loss is much higher than the train loss and the validation top 5 accuracy is much lower than the train top 5 accuracy. This is due to the exposure bias problem which is caused due to the mismatch in data distribution that the model is exposed to during training and inference. While training with teacher forcing the model is exposed to the ground truth captions at every point and never learns to recover from its mistakes. However, during inference the model uses its own outputs to generate the next word and if it makes a mistake then a context is created to which the model was never exposed to during training and the error gets compounded resulting in poor performance at inference. On the other hand, when training without teacher forcing the data distribution that the model is exposed to during training is similar to that at inference and the model learns to recover from its own mistakes during training. Hence at inference time it does not perform as poorly.

Moreover, comparing Tables 2 and 3 it can be observed that in the original study (Ramos et al., 2024), the LSTM decoder performs much better in terms of BLEU scores when trained without teacher forcing as compared to when trained with teacher forcing. However, it is important to note that in their codebase, there no implementation of training without teacher forcing and inference was done with teacher forcing which is incorrect. This study fixes those issues and Table 3 shows that training without teacher forcing results in lower BLEU scores as compared to training with teacher forcing. This can be explained due to slow convergence caused by gradient instability since the model relies on its own output to predict the next word. An error in prediction can be compounded for each subsequent word resulting in gradients being updated in the wrong direction and the model taking time to converge resulting in less accurate captions and lower BLEU scores. Whereas in the case of teacher forcing, the decoder is given the correct word at each time point resulting in stable, faster convergence and higher BLEU scores.



**Figure 1.** BLEU-4 curves for LSTM and transformer decoders training with and without TF

Figure 1 backs the argument since decoders trained without teacher forcing display slow convergence and lower BLEU-4 scores as compared to training with teacher forcing. The experiment ran till 90 epochs for both decoders after which it was timed-out by the system since training without teacher forcing requires waiting for the model’s output at each step which is time-consuming. However, the models were yet to converge as they did not stop due to early stopping and showed small improvements after every other epoch. On the other hand, in the case of training with teacher forcing for both decoders majority of the improvement in BLEU-4 scores took place till epochs 20-30 after which they reached convergence. The LSTM decoder stopped at epoch 53 due to early stopping whereas the transformer decoder showed small improvements till epoch 120.

To qualitatively analyze which training strategy gives more accurate captions, the best checkpoints of each model and strategy are used to generate captions of an unseen image and are compared.

|  |  |  |
| --- | --- | --- |
|  | **Decoder + Strategy** | **Generated Caption** |
| (LSTM + Att) + No TF | A dog sitting a a bench a a a bench |
| Transformer + No TF | A white dog a a a a a a |
| (LSTM + Att) + TF | A couple of dogs sitting on a bench |
| Transformer + TF | A white dog is sitting on a bench |
| True Captions | 1. A large white dog is sitting on a bench beside an elderly man  2. A large white dog sits on a bench with people next to a path  3. A large dog sits just his bottom on a park bench  4. A dog sitting on a bench next to an old man  5. A couple of people sitting on a bench next to a dog |

**Table 4.** Captions generated by LSTM and transformer decoder with and without TF

Table 4 shows that the captions generated by both decoders trained without teacher forcing do not make sense grammatically and are incomplete sentences. This relates to the fact that without teacher forcing, the models have unstable training and low BLEU scores resulting in poor captions. However, with teacher forcing both decoders generate grammatically correct and complete captions since they are able to achieve convergence and display higher BLEU scores. The LSTM decoder generates a caption that is partially correct whereas the transformer decoder generates a correct caption which can be further detailed.

Thus, for both decoders, although training with teacher forcing results in exposure bias, it gives higher BLEU scores and more accurate captions which is the more appropriate metric for NLP tasks hence it is selected as the training strategy for further experiments.

### 4.2.2. Selecting the Best Decoder

The core objective of this study was to implement a transformer decoder that is able to capture both local and global context from the image features and incorporate them accurately in the generated captions which would be compared to the LSTM + attention decoder in the original study (Ramos et al., 2024). For this purpose, both decoders were built and integrated separately in the image captioning architecture with ConvNeXt and trained using the same conditions. In section 4.2.1, it was observed that training with teacher forcing generates better results hence both decoders were trained with teacher forcing and their best checkpoints based on the validation BLEU-4 score was saved and tested on an unseen test set. The results are presented in Table 5.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Decoder** | **Test Loss** | **Test top5 Acc** | **Bleu-1** | **Bleu-2** | **Bleu-3** | **Bleu-4** |
| **LSTM + Att** | 7.39 | 31.94 | 76.48 | 57.83 | 43.21 | 31.66 |
| **Transformer** | 8.10 | 31.85 | 75.96 | 57.61 | 43.51 | **32.34** |

**Table 5.** Test metrics of LSTM and Transformer decoders

According to the results, it can be seen that the LSTM decoder performs slightly better in terms of test loss and top 5 accuracy making it a slightly more confident model. The BLEU scores are quite similar for both decoders. BLEU scores are more relevant metrics for NLP tasks and since BLEU-4 captures the highest n-gram overlap, it is the deciding metric for generating captions similar to the true captions. Since the transformer decoder has a higher BLEU-4 score of 32.34, it is selected as the decoder for further experiments. This supports the original hypothesis of the study which stated that since transformers process the image features and previously generated words in parallel to generate the next word, they are able to capture both local and global contexts allowing them to generate more accurate captions. However, the LSTM decoder has similar performance due to its integrated attention module which helps it to focus on relevant parts of the image features while generating each word providing it with the appropriate context. The theme of the transformer decoder outperforming the LSTM decoder in terms of BLEU-4 scores during training and validation as well can be observed in Table 3 and Figure 1.

## 4.3. The Impact of Finetuning ConvNeXt

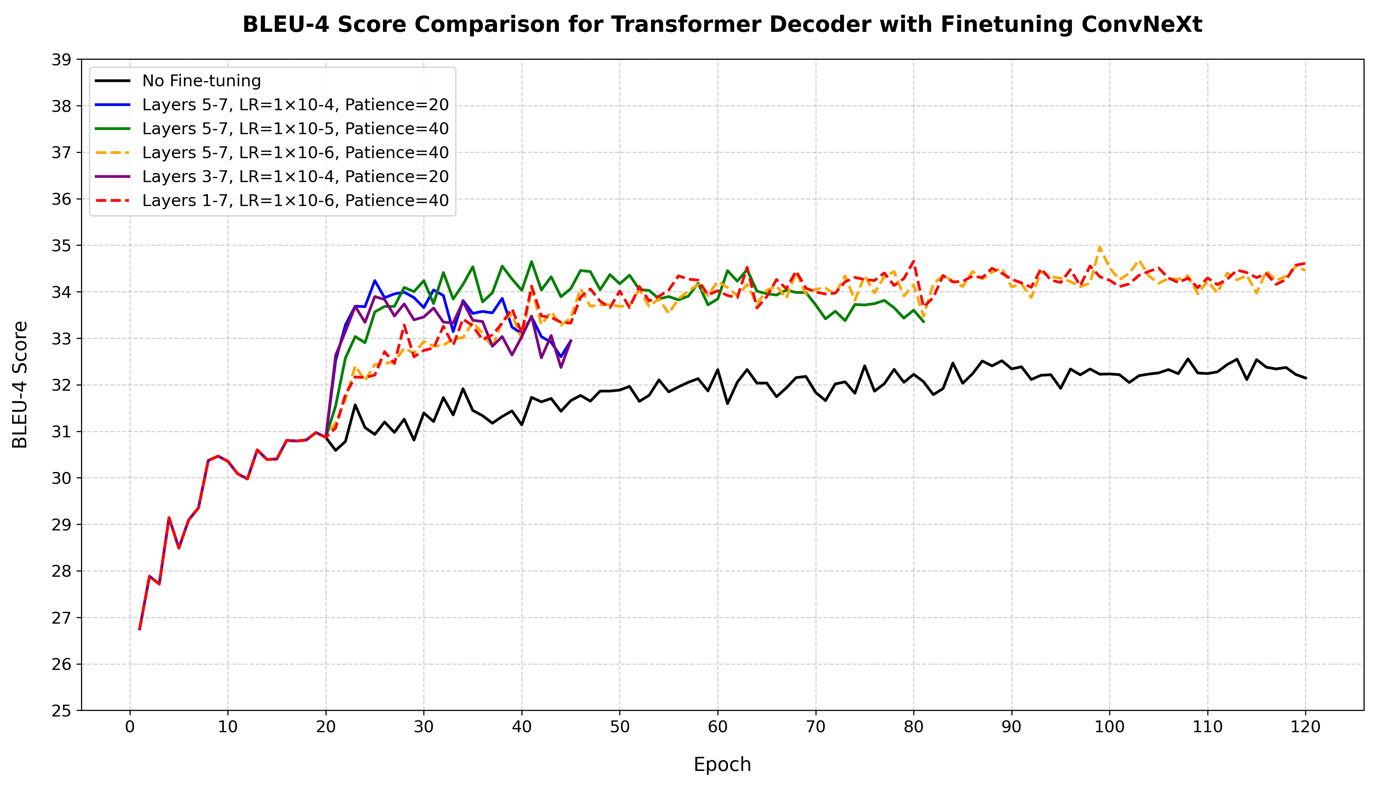
### 4.3.1. Quantitative Analysis

In the next set of experiments, different layers of the ConvNeXt are also finetuned to investigate whether updating the weights for the image captioning task improves the quality of generated captions and if so then what are the optimal layers to finetune. As mentioned earlier, these experiments are done using the transformer decoder trained with teacher forcing. Apart from the layers being finetuned, the learning rate along with the patience for early stopping are adjusted for smaller more gradual updates. The ConvNeXt is frozen for the first 20 epochs to avoid early gradients from the transformer decoder corrupting the pretrained weights.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Layers Finetuned** | **Learning Rate** | **Patience (epochs)** | **Val loss** | **Val top5 acc** | **Bleu-1** | **Bleu-2** | **Bleu-3** | **Bleu-4** |
| None | - | 20 | 8.10 | 31.61 | 76.16 | 58.01 | 43.86 | 32.56 |
| 5 - 7 | 1×10-4 | 20 | 7.76 | 32.63 | 77.74 | 60.04 | 45.77 | 34.24 |
| 5 - 7 | 1×10-5 | 40 | 7.96 | 32.72 | 78.02 | 60.53 | 46.28 | 34.65 |
| 5 - 7 | 1×10-6 | 40 | 8.11 | 32.39 | 77.90 | 60.38 | 46.32 | **34.96** |
| 3 - 7 | 1×10-4 | 20 | 7.77 | 32.52 | 77.52 | 59.84 | 45.50 | 33.90 |
| 1 - 7 | 1×10-6 | 40 | 7.96 | 32.37 | 77.93 | 60.37 | 46.18 | 34.66 |

**Table 5.** Validation metrics of finetuning different layers of ConvNeXt

According to Table 5, it can be seen that finetuning the ConvNeXt does not improve validation loss and top 5 accuracy by much since they are limited by the exposure bias problem however it does improve the model’s performance in terms of BLEU scores. It is interesting to see that finetuning deeper layers does not necessarily improve performance since finetuning layers 5-7 has a slightly better performance in terms of BLEU scores as compared to finetuning layers 3-7 and layers 1-7 while keeping the learning rate and patience constant. This proves that for the task of image captioning, finetuning later layers that are responsible for more complex features is enough since earlier layers are responsible for simple features like edges and lines which are consistent for any visual task hence finetuning them has no added benefit. Moreover, decreasing the learning rate and increasing the patience for early stopping allows the model to make smaller updates to the pretrained weights preventing from corrupting them and making their values move in the right direction. Due to these reasons, finetuning layers 5-7 with a learning rate of 1×10-6 and patience of 40 epochs allows the architecture to achieve the highest BLEU-4 score of 34.96 which outperforms the original study (Ramos et al., 2024) that achieved the highest score of 34.76.



**Figure 2.** BLEU-4 curves with finetuning different layers of the ConvNeXt

Figure 2 shows that all the line graphs are identical for the first 20 epochs since the ConvNeXt is frozen. After epoch 20, the line graph when there is no finetuning has the least improvement across all epochs which supports the argument that finetuning the weights for the image captioning task does result in an improvement in BLEU scores. Finetuning layers 5-7 and 3-7 with learning rate 1×10-4 and patience 20 shows a steep improvement initially till epoch 24 followed by a sharp decline and then stopping early at epoch 44 suggesting that the ConvNeXt’s pretrained weights might have been updated too aggressively resulting in them entering a non-optimal space and not being able to recover. Finetuning layers 5-7 with a slightly lower learning rate of 1×10-5 and patience of 40 epochs shows early signs of improvement as well which then flattens out and then gradually declines coming to an early stop around epoch 80. However, finetuning layers 5-7 and 1-7 with a very low learning rate of 1×10-6 and patience of 40 epochs shows steady improvements after 20 epochs which continues till epoch 120 and manages to achieve the highest BLEU-4 scores. This shows that small and gradual updates to the pretrained weights results in them moving in the right direction towards an optimal space without getting corrupted.

### 4.3.2. Qualitative Analysis

For each configuration of finetuning, the checkpoint that has the highest validation BLEU-4 score during training is used to generate a caption for an unseen image. These captions are then compared to carry out a qualitative analysis to assess which configuration generates the most accurate captions and whether the BLEU-4 scores translate to the actual caption quality.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Layers Finetuned** | **Learning Rate** | **Patience** | **Generated Caption** |
| None | - | 20 | A white dog is sitting on a bench |
| 5 - 7 | 1×10-4 | 20 | A man and woman sitting on a bench with a dog |
| 5 - 7 | 1×10-5 | 40 | A man sitting on a bench with a dog |
| 5 - 7 | 1×10-6 | 40 | A white dog standing next to a park bench |
| 3 - 7 | 1×10-4 | 20 | A couple of people that are sitting on a bench |
| 1 - 7 | 1×10-6 | 40 | A white dog standing next to a park bench |

**Table 6.** Captions generated by finetuning different layers of the ConvNeXt

Observing the generated captions in Table 6, it can be seen that when the ConvNeXt is not finetuned, a relatively accurate caption is generated however it lacks detail. However, after finetuning the ConvNeXt, the generated captions contain some more detail as they mention the man, woman or people sitting on the bench as well. From Table 4, it can be seen that these details are mentioned in the true captions which explains the higher BLEU scores after finetuning. Finetuning layers 5-7 or layers 1-7 with the lowest learning rate of 1×10-6 and patience 40 epochs, generate the same caption which does not talk about the people on the bench but captures the detail that the bench is in a park. This is also mentioned in one of the true captions. The quantitative analysis shows that higher BLEU scores do not necessarily mean more accurate captions however, it is important to note that this is the case for a single image and cannot be generalized to all images.

## 4.4. The Impact of Pretrained Word Embeddings

# Discussion, Reflection and Conclusion

# Glossary

# References

# Appendices and additional files